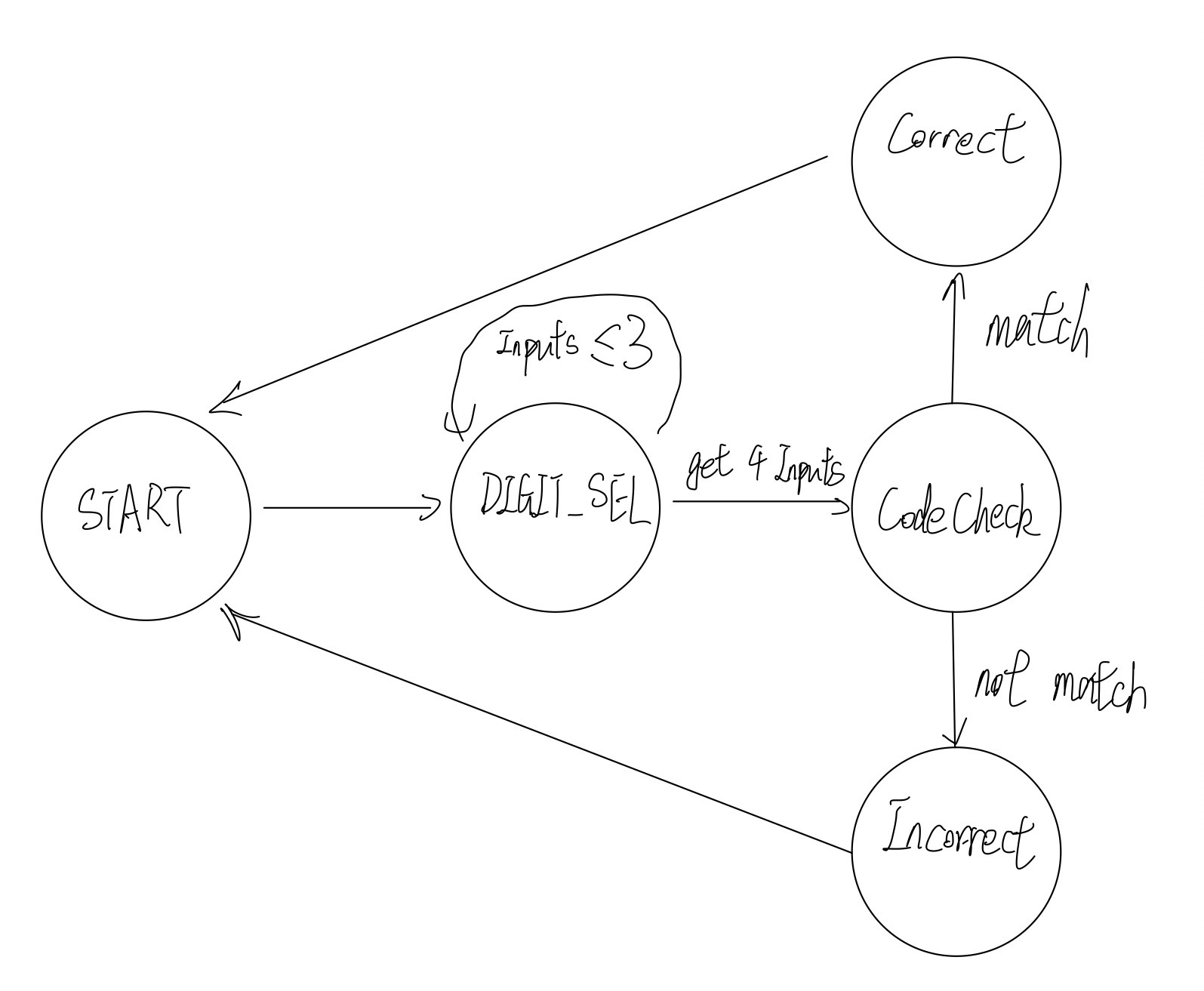
Hongyu Zeng

Ethan

Professor Beichel

Lab3 Report

1. **Introduction**
2. **Schematics**



1. **Discussion**

Things we’ve tested

|  |  |
| --- | --- |
| Based on rotating direction, 7-seg increase (cw) or decrease (ccw) |  |
| Until the rotation is completed, digit won’t change |  |
| Long press reset the code |  |
| Press shorter than 1s remembers the code |  |
| Press longer than 1s but shorter than 3s does nothing |  |
| If password matches, LED is turn on for 5s, lock input for 5s |  |
| If password doesn’t match, display “\_” and lock input for 12s |  |
|  |  |

There are 5 states for this program.

**Start State:**

Turn off LED, reset the next storage register for input, reset codes, turn off LED, display “-”. It goes right to the digit\_sel no matter what.

**Digit\_sel:**

Repeatedly call debounce for rpg to get PIND 7 and 6’s values. The debounce for rpg samples 9 times and based on the number of 0s, it decides the final value for A and B and store them in r25. Since the detent value is 11, if the rpg starts rotating, then one of them will change and it enters the rotating subroutine, change the r29’s bit 3 or 2 to record the rotating direction in the rotating subroutine based on the value of r25, and constantly debounce the rpg and exits the loop only if the rpg reaches the next detent position.

Then the buttonCheck will do button debounce (buttonDeb) and change the bit 7 of r29 to indicate the state of the button (1 for pressed). The debounce is like rpg’s. If the button is not pressed then go back to digit\_sel main loop. Else if the button is pressed, go the button\_pressed loop. Based on the time being pressed (it enters a loop until release) it will have the following effects: 1. less than 1s, load the bit map register r16 to the storage register and update the next storage register. 2. more than 3s, it will exit the loop without need for release and reset the code like in Start State. I choose Z pair and store 3000 to it. Every time the loop button\_pressed is iterated, it will take about 1ms (mainly from the buttonDeb subroutine). So if the Z reaches 0, it will branch to reset the code; if the next button debounce shows that it is no long pressed while Z is still positive, then if the R31 is larger or equal to 111 (Z is >= 1792) then it means that the button is pressed less than 1.2s, which goes to memorize\_code subroutine. Otherwise just do nothing and goes back to the main loop of digit\_sel. FYI: When the button is pressed user won’t be able to change the digit using rpg.

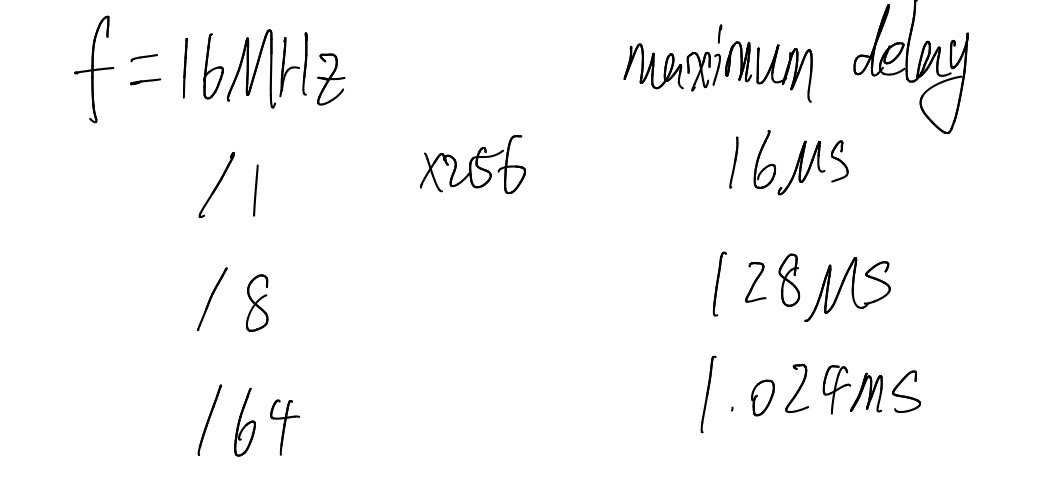
Then the Digit\_sel call the subroutine “checkInputFull” to check if the number of inputs is 4 based on the bit 5 on the r29 (my customized flag register), go to the checkcode state or keep looping in digit\_sel.

**CodeCheck:** Compare the r18-r21's value with A, 6, 7, 8 one by one. If any of them doesn’t match, go to incorrect state, else go to correct state

**IncorrectState:** load r16 with the corresponding bit map for lower bar, stuck in the state for 12s and goes back to start state.

**CorrectState:** make the PORTB5 equals to 1 to light up the LED for 5s then go back to start state

The timer I used most is the delay\_100us\_TC0 which utilized interal clocks for the delay. We are using normal mode. By setting prescaler to /8, the maximum timer and having system clock frequency as 16MHz, (as this has the maximum cycle of 128us) I set the counter to 56 so it has 200 iteration per delay then we have a 100 us delay. This delay subroutine is also called by delay (Z times the 100us), delay\_oms(1ms), delay\_long(Z times 1ms) for longer delays.



So why 0.1 ms/100us? The minimum phase difference is 2.5ms. My debounce for rpg and button both take 9 samples so the time for debounces are about 1ms which intotal is about 2ms. This makes sure that I don’t skip any phase for rpg.

1. **Conclusion**

The rpg debounce is similar to button debounce. By doing this lab, we are familiar with how RPG works and how to detect rotating direction based on the A/B’s values.

1. **Appendix**

Source Code